**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 14AB**

# Lab 05: tesT first programming - i

**CLO-03:** Design and develop solutions based on Software Construction principles.  
**CLO-04:** Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Date: 06th Oct 2025**

**Time: 09:00 AM** **- 11:50 PM   
 02:00 PM – 04:50 PM**

**Submitted By: Sikander Hayat Khan**

**CMS ID: 464544**

**GitHub:** [**https://github.com/Sikander-Hayat-Khan/SC-LAB-5-TFP-1.git**](https://github.com/Sikander-Hayat-Khan/SC-LAB-5-TFP-1.git)

**Instructor: Dr. Sidra Sultana  
Lab Engineer: Mr. Aftab Farooq**

# Lab 05: Test- First Programming: Tweet Tweet

## Introduction:

Students will have hands-on experience of **test-first programming**. Given a set of specifications, you will write **unit tests** that check for compliance with the **specifications**, and then **implement** code that meets the specifications.

**Material:** https://ocw.mit.edu/ans7870/6/6.005/s16/psets/ps1/

Lectures on LMS regarding **designing Specifications** and **Testing**

## Lab Tasks:

Solve problem 1 and 2 of problem set 1 listed on the link. The goal of the problem set is tto build a toolbox of methods that can extract information from a set of tweets downloaded from Twitter.

### Test-First Programming:

1. Study the specification of the method carefully.
2. Write JUnit tests for the method according to the spec.
3. Implement the method according to the spec.
4. Revise your implementation and improve your test cases until your implementation passes all your tests.

## Task1: Extracting data from Tweets

In this problem, you will test and implement the methods in **Extract.java**. You’ll find **Extract.java** in the **src** folder, and a JUnit test class **ExtractTest.java** in the test folder. Separating implementation code from test code is a common practice in development projects. It makes the implementation code easier to understand, uncluttered by tests, and easier to package up for release

* 1. Devise, document, and implement test cases for **getTimespan()** and **getMentionedUsers()** , and put them in **ExtractTest.java .**
  2. Implement **getTimespan()** and **getMentionedUsers()** , and make sure your tests pass.

If you want to see your code work on a live sample of tweets, you can run **Main.java** . ( Main.java will not be used in grading, and you are free to edit it as you wish.)

### Hints:

* Note that we use the class **Instant** to represent the date and time of tweets. You can check **this article on Java 8 dates and times** to learn how to use **Instant** .
* You may wonder what to do about lowercase and uppercase in the return value of **getMentionedUsers()** . This spec has an underdetermined postcondition, so read the spec carefully and think about what that means for your implementation and your test cases.
* **getTimespan()** *also* has an underdetermined postcondition in some circumstances, which gives the implementor (you) more freedom and the client (also you, when you’re writing tests) less certainty about what it will return.
* Read the spec for the **Timespan** class carefully, because it may answer many of the questions you have about **getTimespan()** .

## Task2: Filtering lists of Tweets

In this problem, you will test and implement the methods in **Filter.java** .

1. Devise, document, and implement test cases for **writtenBy()** , **inTimespan()** , and **containing()** , and put them in **FilterTest.java** .
2. Implement **writtenBy()** , **inTimespan()** , and **containing()** , and make sure your tests pass.

Hints:

* For questions about lowercase/uppercase and how to interpret timespans, reread the hints in the previous question.
* For all problems on this problem set, you are free to rewrite or replace the provided example tests and their assertions.

|  |
| --- |
| **Solution** |
| **Task 1**  **Code**  **Extract.java**  package twitter;  import java.time.Instant; import java.util.\*; import java.util.regex.\*;  // Extract consists of methods that extract information from a list of tweets. public class Extract {   */\*\*  \* Get the time period spanned by tweets.  \*/* public static Timespan getTimespan(List<Tweet> tweets) {  if (tweets.isEmpty()) {  // The spec doesn't define behavior for empty input; choose a safe default  Instant now = Instant.*now*();  return new Timespan(now, now);  }   Instant start = tweets.get(0).getTimestamp();  Instant end = start;   for (Tweet t : tweets) {  Instant time = t.getTimestamp();  if (time.isBefore(start)) start = time;  if (time.isAfter(end)) end = time;  }   return new Timespan(start, end);  }   */\*\*  \* Get usernames mentioned in a list of tweets.  \*/* public static Set<String> getMentionedUsers(List<Tweet> tweets) {  Set<String> users = new HashSet<>();   // Regex explanation:  // (?<=^|[^A-Za-z0-9\_]) ensures '@' is not part of another word (not preceded by a valid username char)  // @([A-Za-z0-9\_]+) captures the username part  // \b ensures it ends cleanly (not followed by another username-valid char)  Pattern mentionPattern = Pattern.*compile*("(?<=^|[^A-Za-z0-9\_])@([A-Za-z0-9\_]+)\\b");   for (Tweet t : tweets) {  Matcher matcher = mentionPattern.matcher(t.getText());  while (matcher.find()) {  String username = matcher.group(1).toLowerCase();  users.add(username);  }  }   return users;  }  }  **ExtractTest.java**  package twitter;  import static org.junit.Assert.\*;  import java.time.Instant; import java.util.\*;  import org.junit.Test;  public class ExtractTest {   /\*  \* Testing strategy  \*  \* getTimespan(tweets):  \* - tweets.size() = 0, 1, >1  \* - timestamps in order, reverse order, random order  \*  \* getMentionedUsers(tweets):  \* - no mentions  \* - one mention at start/middle/end of text  \* - multiple mentions  \* - case-insensitive duplicates (e.g., @Bob, @BOB)  \* - invalid mentions inside email (e.g., bit@mit.edu)  \*/   private static final Instant *d1* = Instant.*parse*("2016-02-17T10:00:00Z");  private static final Instant *d2* = Instant.*parse*("2016-02-17T11:00:00Z");  private static final Instant *d3* = Instant.*parse*("2016-02-17T12:00:00Z");   private static final Tweet *tweet1* = new Tweet(1, "alyssa", "is it reasonable to talk about rivest so much?", *d1*);  private static final Tweet *tweet2* = new Tweet(2, "bbitdiddle", "rivest talk in 30 minutes #hype", *d2*);  private static final Tweet *tweet3* = new Tweet(3, "charlie", "@Alice and @BOB are attending the talk!", *d3*);   @Test(expected=AssertionError.class)  public void testAssertionsEnabled() {  assert false; // make sure assertions are enabled with VM argument: -ea  }   // ---- getTimespan() tests ----   @Test  public void testGetTimespanTwoTweets() {  Timespan timespan = Extract.*getTimespan*(Arrays.*asList*(*tweet1*, *tweet2*));   *assertEquals*("expected start", *d1*, timespan.getStart());  *assertEquals*("expected end", *d2*, timespan.getEnd());  }   @Test  public void testGetTimespanOneTweet() {  Timespan timespan = Extract.*getTimespan*(Collections.*singletonList*(*tweet1*));  *assertEquals*(*d1*, timespan.getStart());  *assertEquals*(*d1*, timespan.getEnd());  }   @Test  public void testGetTimespanMultipleUnorderedTweets() {  Timespan timespan = Extract.*getTimespan*(Arrays.*asList*(*tweet3*, *tweet1*, *tweet2*));  *assertEquals*("earliest start", *d1*, timespan.getStart());  *assertEquals*("latest end", *d3*, timespan.getEnd());  }   // ---- getMentionedUsers() tests ----   @Test  public void testGetMentionedUsersNoMention() {  Set<String> mentionedUsers = Extract.*getMentionedUsers*(Arrays.*asList*(*tweet1*));  *assertTrue*("expected empty set", mentionedUsers.isEmpty());  }   @Test  public void testGetMentionedUsersSingleMention() {  Tweet tweet = new Tweet(4, "bob", "Hello @Alice", *d1*);  Set<String> mentions = Extract.*getMentionedUsers*(Arrays.*asList*(tweet));  *assertEquals*(Set.*of*("alice"), mentions);  }   @Test  public void testGetMentionedUsersCaseInsensitiveDuplicates() {  Tweet t1 = new Tweet(5, "user", "Hey @BOB", *d1*);  Tweet t2 = new Tweet(6, "user", "@bob good morning", *d2*);  Set<String> mentions = Extract.*getMentionedUsers*(Arrays.*asList*(t1, t2));  *assertEquals*("mentions should be case-insensitive", Set.*of*("bob"), mentions);  }   @Test  public void testGetMentionedUsersIgnoresEmails() {  Tweet tweet = new Tweet(7, "user", "contact me at example@mit.edu", *d1*);  Set<String> mentions = Extract.*getMentionedUsers*(Arrays.*asList*(tweet));  *assertTrue*("email address should not be treated as mention", mentions.isEmpty());  }   @Test  public void testGetMentionedUsersMultipleMentions() {  Set<String> mentions = Extract.*getMentionedUsers*(Arrays.*asList*(*tweet3*));  *assertTrue*(mentions.contains("alice"));  *assertTrue*(mentions.contains("bob"));  *assertEquals*("expected two unique mentions", 2, mentions.size());  } }  **Screenshot**    **Task 2**  **Code**  **Filter.java**  package twitter;  import java.time.Instant; import java.util.\*; import java.util.stream.Collectors;  public class Filter {   */\*\*  \* Find tweets written by a particular user.  \*/* public static List<Tweet> writtenBy(List<Tweet> tweets, String username) {  List<Tweet> result = new ArrayList<>();  for (Tweet t : tweets) {  if (t.getAuthor().equalsIgnoreCase(username)) {  result.add(t);  }  }  return result;  }   */\*\*  \* Find tweets that were sent during a particular timespan.  \*/* public static List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan) {  Instant start = timespan.getStart();  Instant end = timespan.getEnd();  List<Tweet> result = new ArrayList<>();   for (Tweet t : tweets) {  Instant time = t.getTimestamp();  if ((time.equals(start) || time.isAfter(start)) &&  (time.equals(end) || time.isBefore(end))) {  result.add(t);  }  }  return result;  }   */\*\*  \* Find tweets that contain certain words.  \*/* public static List<Tweet> containing(List<Tweet> tweets, List<String> words) {  List<Tweet> result = new ArrayList<>();  if (words.isEmpty()) return result;   // lowercase all search words for case-insensitive comparison  Set<String> lowerWords = words.stream()  .map(String::toLowerCase)  .collect(Collectors.*toSet*());   for (Tweet t : tweets) {  String[] tweetWords = t.getText().split("\\s+");  for (String w : tweetWords) {  String cleaned = w.replaceAll("[^A-Za-z0-9#@]", "").toLowerCase();  if (lowerWords.contains(cleaned)) {  result.add(t);  break; // add each tweet only once  }  }  }  return result;  } }  **FilterTest.java**  package twitter;  import static org.junit.Assert.\*;  import java.time.Instant; import java.util.\*;  import org.junit.Test;  public class FilterTest {   /\*  \* Testing strategy  \*  \* writtenBy(tweets, username):  \* - tweets.size() = 0, 1, >1  \* - username case = same, different case  \* - author present or absent  \*  \* inTimespan(tweets, timespan):  \* - tweet before, within, after timespan  \* - boundary cases (exactly at start/end)  \*  \* containing(tweets, words):  \* - no words  \* - one word match  \* - multiple words  \* - case-insensitive matching  \* - no tweet matches  \*/   private static final Instant *d1* = Instant.*parse*("2016-02-17T10:00:00Z");  private static final Instant *d2* = Instant.*parse*("2016-02-17T11:00:00Z");  private static final Instant *d3* = Instant.*parse*("2016-02-17T12:00:00Z");   private static final Tweet *tweet1* = new Tweet(1, "alyssa", "is it reasonable to talk about rivest so much?", *d1*);  private static final Tweet *tweet2* = new Tweet(2, "bbitdiddle", "rivest talk in 30 minutes #hype", *d2*);  private static final Tweet *tweet3* = new Tweet(3, "alyssa", "Lunch time soon", *d3*);   @Test(expected=AssertionError.class)  public void testAssertionsEnabled() {  assert false;  }   // ----- writtenBy() tests -----   @Test  public void testWrittenBySingleResult() {  List<Tweet> writtenBy = Filter.*writtenBy*(Arrays.*asList*(*tweet1*, *tweet2*), "alyssa");   *assertEquals*("expected singleton list", 1, writtenBy.size());  *assertTrue*(writtenBy.contains(*tweet1*));  }   @Test  public void testWrittenByCaseInsensitive() {  List<Tweet> writtenBy = Filter.*writtenBy*(Arrays.*asList*(*tweet1*, *tweet2*, *tweet3*), "ALYSSA");   *assertEquals*("expected two tweets by alyssa", 2, writtenBy.size());  *assertTrue*(writtenBy.contains(*tweet1*));  *assertTrue*(writtenBy.contains(*tweet3*));  }   @Test  public void testWrittenByNoMatch() {  List<Tweet> writtenBy = Filter.*writtenBy*(Arrays.*asList*(*tweet1*, *tweet2*), "unknown");  *assertTrue*("expected empty list", writtenBy.isEmpty());  }   // ----- inTimespan() tests -----   @Test  public void testInTimespanInclusiveBoundaries() {  Instant start = *d1*;  Instant end = *d2*;  Timespan span = new Timespan(start, end);   List<Tweet> inSpan = Filter.*inTimespan*(Arrays.*asList*(*tweet1*, *tweet2*, *tweet3*), span);  *assertTrue*("should include tweet at start", inSpan.contains(*tweet1*));  *assertTrue*("should include tweet at end", inSpan.contains(*tweet2*));  *assertFalse*("should exclude tweet after end", inSpan.contains(*tweet3*));  }   @Test  public void testInTimespanNoTweetsInRange() {  Instant start = Instant.*parse*("2016-02-17T13:00:00Z");  Instant end = Instant.*parse*("2016-02-17T14:00:00Z");  Timespan span = new Timespan(start, end);   List<Tweet> inSpan = Filter.*inTimespan*(Arrays.*asList*(*tweet1*, *tweet2*, *tweet3*), span);  *assertTrue*("expected empty list", inSpan.isEmpty());  }   // ----- containing() tests -----   @Test  public void testContainingSingleWord() {  List<Tweet> containing = Filter.*containing*(Arrays.*asList*(*tweet1*, *tweet2*, *tweet3*), Arrays.*asList*("talk"));  *assertTrue*("expected tweet1 and tweet2", containing.containsAll(Arrays.*asList*(*tweet1*, *tweet2*)));  *assertFalse*("tweet3 does not contain 'talk'", containing.contains(*tweet3*));  }   @Test  public void testContainingCaseInsensitive() {  List<Tweet> containing = Filter.*containing*(Arrays.*asList*(*tweet1*, *tweet2*), Arrays.*asList*("RIVEST"));  *assertTrue*("expected both tweets mentioning rivest", containing.containsAll(Arrays.*asList*(*tweet1*, *tweet2*)));  }   @Test  public void testContainingNoMatches() {  List<Tweet> containing = Filter.*containing*(Arrays.*asList*(*tweet1*, *tweet2*), Arrays.*asList*("banana"));  *assertTrue*("expected empty list", containing.isEmpty());  }   @Test  public void testContainingMultipleWords() {  List<Tweet> containing = Filter.*containing*(Arrays.*asList*(*tweet1*, *tweet2*, *tweet3*), Arrays.*asList*("rivest", "lunch"));  *assertTrue*("expected tweet1 and tweet2 and tweet3", containing.containsAll(Arrays.*asList*(*tweet1*, *tweet2*, *tweet3*)));  } }  **Screenshot** |

Task 1:

Code:

**Source Code: Zip your source code and upload one file per group on LMS as well.**

**Solution**

## Deliverables:

Compile a single word document by filling in the solution part and submit this Word file on LMS. In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to [aftab.farooq@seecs.edu.pk.](mailto:aftab.farooq@seecs.edu.pk.)